Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Computation 197 (2005) 90–121
Information and Computation
www.elsevier.com/locate/ic

Minimal-change integrity maintenance using tuple deletions ☆

Jan Chomicki [a,*], Jerzy Marcinkowski [b]

[a] *Department of Computer Science and Engineering, 201 Bell Hall, University at Buffalo, Buffalo, NY 14260-2000, USA*
[b] *Institute of Informatics, Wroclaw University, Przesmyckiego 20, 51-151 Wroclaw, Poland*

Received 6 December 2002; revised 6 April 2004

Abstract

We address the problem of minimal-change integrity maintenance in the context of integrity constraints in relational databases. We assume that integrity-restoration actions are limited to tuple deletions. We focus on two basic computational issues: *repair checking* (is a database instance a repair of a given database?) and *consistent query answers* [in: ACM Symposium on Principles of Database Systems (PODS), 1999, 68] (is a tuple an answer to a given query in every repair of a given database?). We study the computational complexity of both problems, delineating the boundary between the tractable and the intractable cases. We consider denial constraints, general functional and inclusion dependencies, as well as key and foreign key constraints. Our results shed light on the computational feasibility of minimal-change integrity maintenance. The tractable cases should lead to practical implementations. The intractability results highlight the inherent limitations of any integrity enforcement mechanism, e.g., triggers or referential constraint actions, as a way of performing minimal-change integrity maintenance.

1. Introduction

Inconsistency is a common phenomenon in the database world today. Even though integrity constraints successfully capture data semantics, the actual data in the database often fail to satisfy such

☆ Research supported by NSF Grant IIS-0119186 and UB faculty start-up funds.
* Corresponding author.
E-mail addresses: chomicki@cse.buffalo.edu (J. Chomicki), Jerzy.Marcinkowski@ii.uni.wroc.pl (J. Marcinkowski).

0890-5401/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2004.04.007

constraints. This may happen because the data is drawn from a variety of independent sources (as in data integration [47]) or are involved in complex, long-running activities like workflows.

How to deal with inconsistent data? The traditional way is not to allow the database to become inconsistent by aborting updates or transactions leading to integrity violations. We argue that in present-day applications this scenario is becoming increasingly impractical. First, if a violation occurs because of data from multiple, independent sources being merged [48], there is no single update responsible for the violation. Moreover, the updates have typically already committed. For example, if we know that a person should have a single address but multiple data sources contain different addresses for the same person, it is not clear how to fix this violation through aborting some update. Second, the data may have become inconsistent through the execution of some complex activity and it is no longer possible to trace the cause of the inconsistency to a specific action.

In the context of triggers or referential integrity, more sophisticated methods for handling integrity violations have been developed. For example, instead of being aborted an update may be propagated. In general, the result is at best a consistent database state, typically with no guarantees on its distance from the original, inconsistent state (the research reported in [49] is an exception).

In our opinion, integrity-restoration should be a separate process that is executed after an inconsistency is detected. The restoration should have a minimal impact on the database by trying to preserve as many tuples as possible. This scenario is called from now on *minimal-change integrity maintenance*.

One can interpret the postulate of minimal change in several different ways, depending on whether the information in the database is assumed to be *correct* and *complete*. If the information is complete but not necessarily correct (it may violate integrity constraints), the only way to fix the database is by *deleting* some parts of it. If the information is both incorrect and incomplete, then both insertions and deletions should be considered. In this paper, we focus on the first case. Since we are working in the context of the relational data model, we consider *tuple deletions*. Such a scenario is common in data warehouse applications where dirty data coming from many sources is cleaned in order to be used as a part of the warehouse itself. On the other hand, in some data integration approaches, e.g. [46,47], the completeness assumption is not made. For large classes of constraints, e.g., denial constraints, the restriction to deletions has no impact, since only deletions can remove integrity violations. Another dimension of change minimality is whether updates to selected attributes of tuples are considered as ways to remove integrity violations. We return to the issue of minimal change in Sections 2 and 5.

We claim that a central notion in the context of integrity restoration is that of a *repair* [3]. A repair is a database instance that satisfies the integrity constraints and *minimally* differs from the original database (which may be inconsistent). Because we consider only *deletions of complete tuples* as ways to restore database consistency, the repairs in our framework are *maximal consistent subsets* of the original database instance.

The basic computational problem in this context is *repair checking*, namely checking whether a given database instance is a repair of the original database. The complexity of this problem is studied in the present paper. Typically, repair checking algorithms can be easily adapted to nondeterministically compute repairs (as we show).

Sometimes when the data is retrieved online from multiple, autonomous sources, it is not possible to restore the consistency by constructing a single repair. In that case one has to settle for computing, in response to queries, *consistent query answers* [3], namely answers that are true in every repair of the given database. Such answers constitute a conservative "lower bound" on the information present in